در این پروژه، پردازنده 6 بیتی كه در كلاس طراحي شده را پياده‌سازي كرده و براي آن برنامه‌نويسي مي‌كنيم.

**توجه:** اين پروژه در صورتي قابل قبول است كه براي آن گزارش هم نوشته شود. در اين گزارش نحوه پياده‌سازي پردازنده و اجراي برنامه توسط آن با استفاده از عكس‌هاي مناسب از خروجي شبيه‌سازي نشان داده شود.

**بخش اول (%40 نمره پروژه):** براي انجام اين پروژه ابتدا پردازنده را با استفاده از VHDL يا Verilog پياده‌سازي كرده و صحت عملكرد آن را با اجراي كد زير كه دو عدد 7 و 4 را با هم جمع مي‌كند بررسي كنيد.

LOAD R0, 7

LOAD R1, 4

ADD R0, R1

**بخش دوم (%20 نمره پروژه):** با توجه به اين‌كه اين پردازنده دستور ضرب ندارد، عمل ضرب را با استفاده از عمل جمع و به صورت نرم‌افزاري پياده‌سازي كرده و صحت عملكرد آن را با يك مثال نشان دهيد (مشابه بخش اول یک کد اسمبلی بنویسید که عمل ضرب را انجام دهد). به عنوان مثال، حاصلضرب عدد 8 در 6 را حساب كند.

**بخش سوم (%40 نمره پروژه):** دستور ضرب را با کمترین سربار سخت‌افزاری به مجموعه دستورات اضافه کرده و صحت عملکرد آن را با نوشتن یک کد که حاصلضرب 8 در 6 را حساب كند نشان دهید. توجه کنید که برای این کار نیاز است تغییراتی در سخت‌افزار و کد دستورات ایجاد کنید.

**نمره اضافی (1 نمره):** پیاده‌سازی اسمبلر برای تبدیل کد اسمبلی به کد باینری با استفاده از زبان‌های سطح بالا مانند جاوا و پایتون.

**معماري پردازنده:**



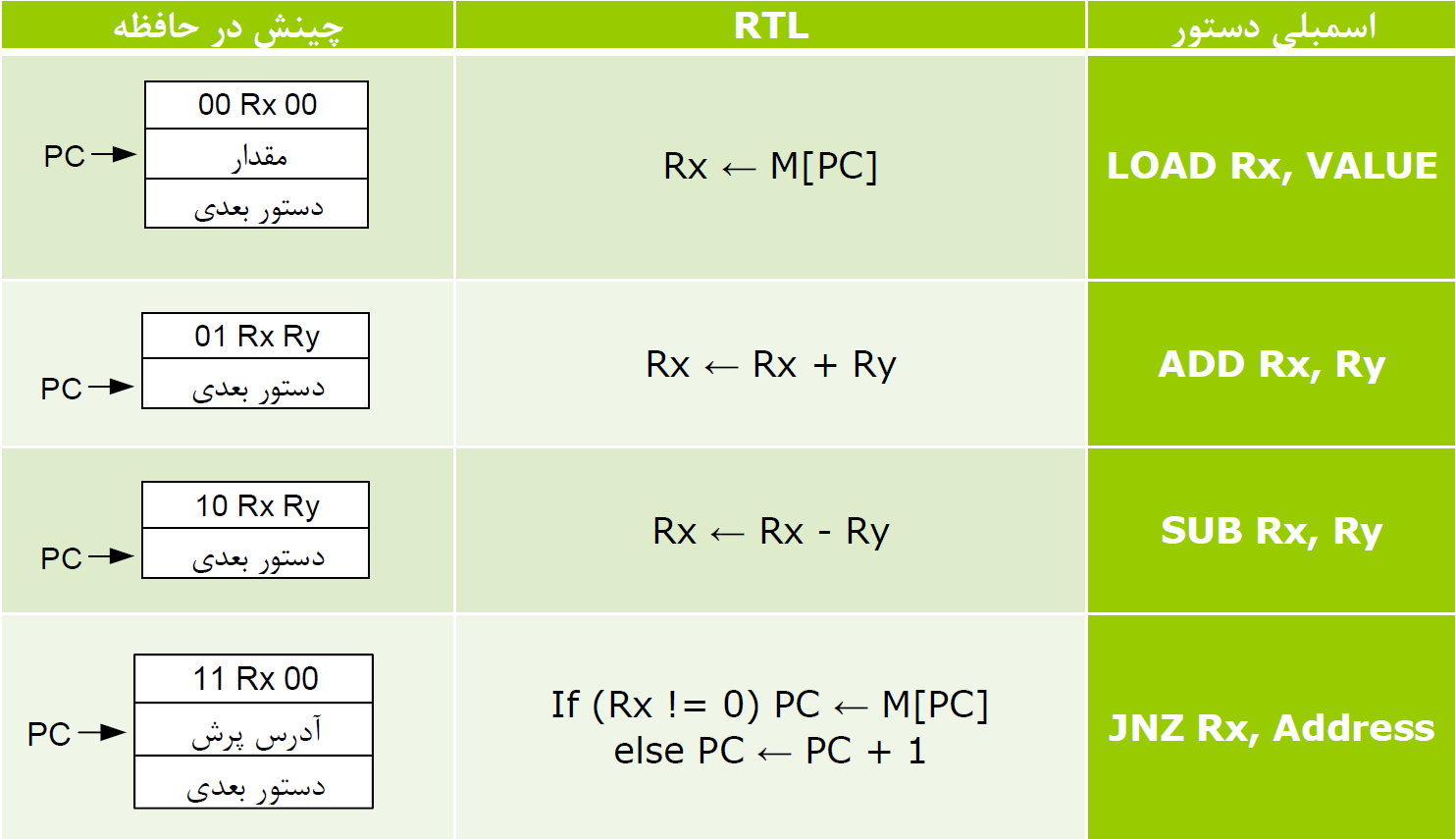
**دستورات پردازنده:**

اين پردازنده چهار دستور LOAD، ADD، SUB و JNZ با كد دستور (Op Code) زير است:

|  |  |
| --- | --- |
| دستور | كد دستور |
| LOAD | 00 |
| ADD | 01 |
| SUB | 10 |
| JNZ | 11 |

**قالب دستورات:**

****



**چارت ASM براي طراحي واحد كنترل:**



به نام خدا

گزارش پروژه پایانی همطراحی

امین دائم دوست

980122680040

محمد حسین رحیمی

980122680036

پوریا عباسی شنبه بازاری

980122681012

توضیحات کلی:

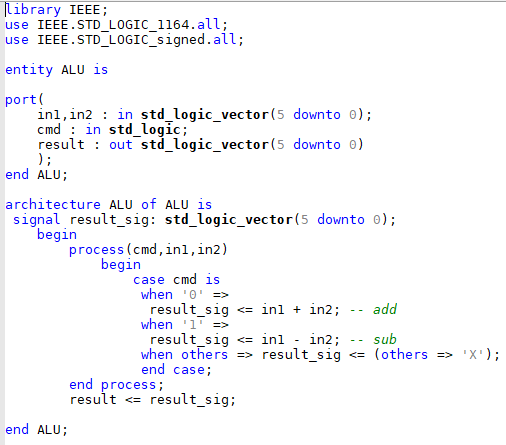
در این پروژه دو پردازده‌ی design1 و design2 طراحی شده اند که اولی برای چهار عملیات ADD، SUB، JNZ، LOAD برای بخش اول پروژه و دومی برای بخش سوم بوده که عملیات MULT را نیز انجام میدهد.

همچنین فایل تبدیل اسمبلی به باینری نیز در پوشه‌ی پروژه قابل مشاهده است که بزای هر کی از پردازنده ها اماده شده است تا دستورات را به صورت مستقیم وارد فایل memory.vhd کند.

همچنین توضیحات opcode ها در بخش اسمبلر با جزئیات نوشته شده.

بخش اول: طراحی پروژه و تست جمع

ALU:



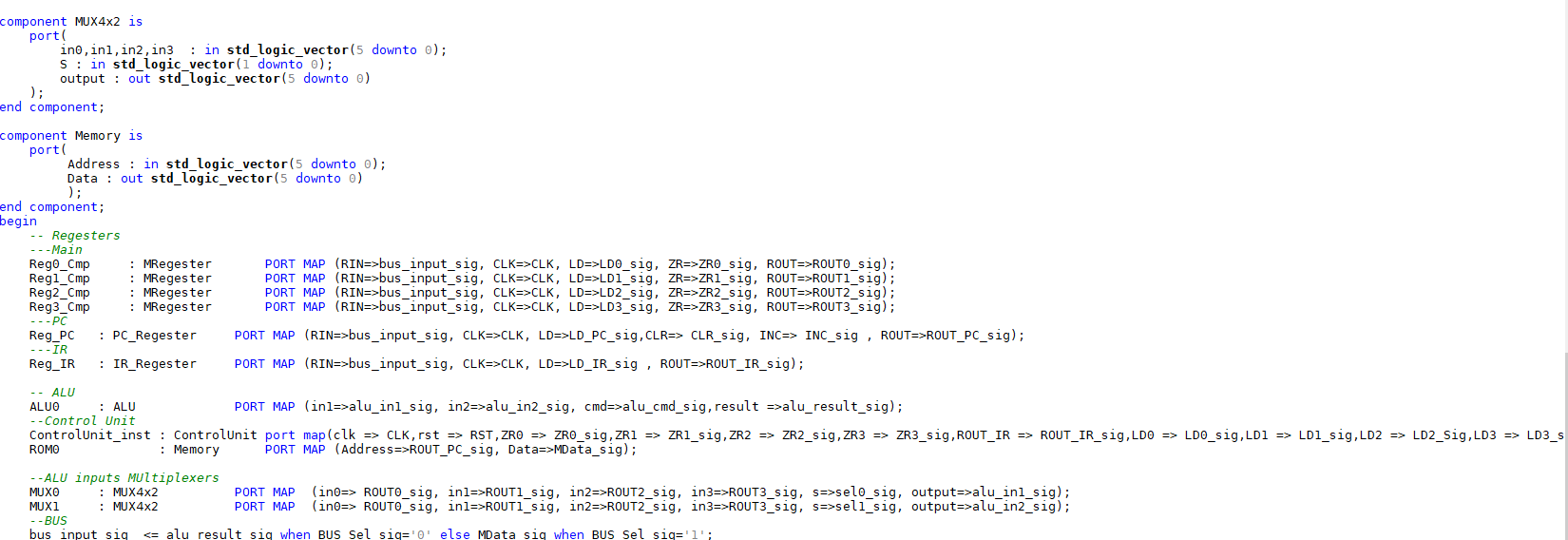
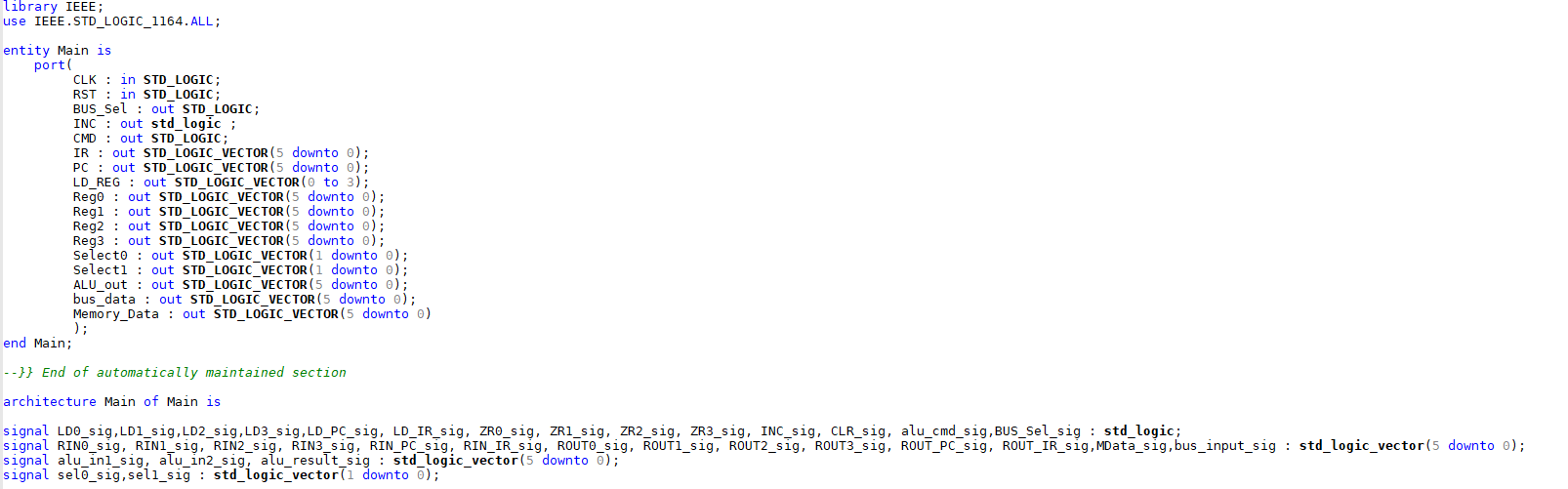
وظیفه این واحد اجرای عمل جمع و تفریق دو رجیستر است، اگر CMD=1 عمل تفریق و اگر CMD=0 عمل جمع را انجام میدهد.

MRegister:



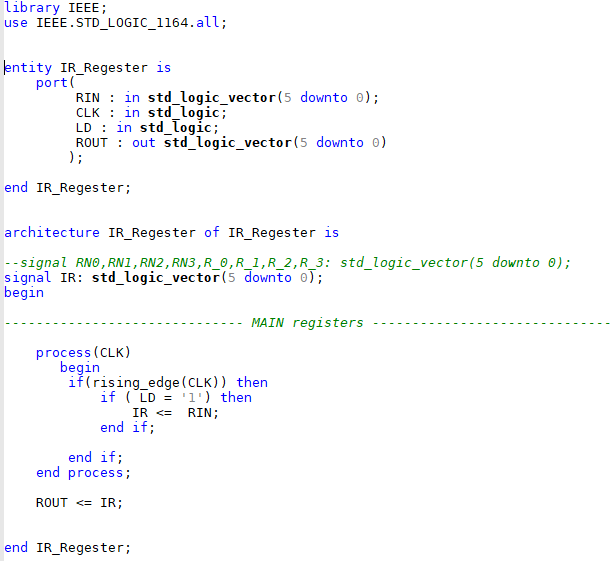
چهار رجیستر اصلی، که مقداری را لود کرده و خروجی میدهند. همچنین اگر برابر 0 باشند، مقدار ZR=1 میکنند.

Main:



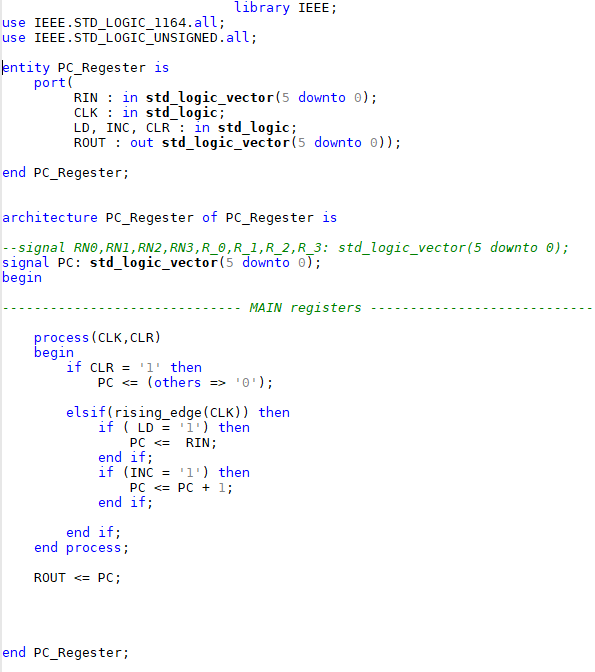
وظیفه تعریف کامپوننت ها و برقراری اتصال بین آن ها و همچنین مپ کردن پورت ها را بر عهده دارد

IR\_register:



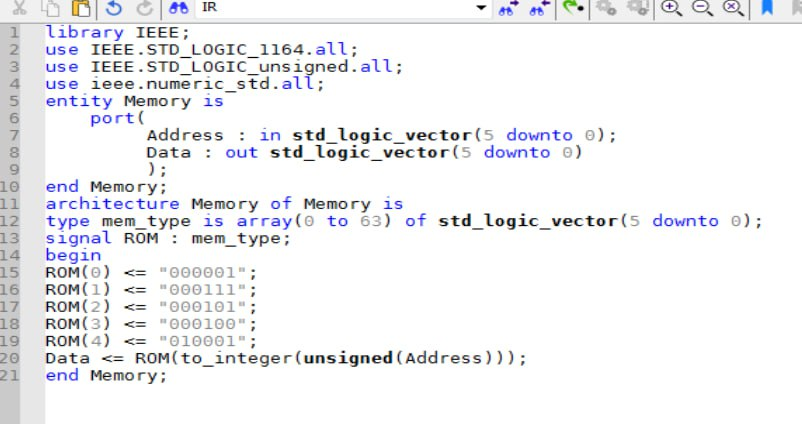
رجیستر Instruction Register که وظیفه خواندن دستور مموری و ارسال آن به واحد کنترل را برعهده دارد

PC\_Register:



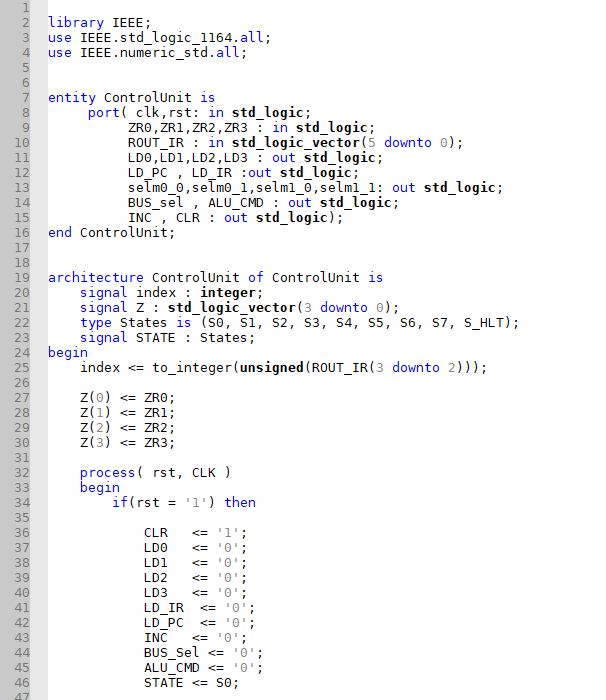
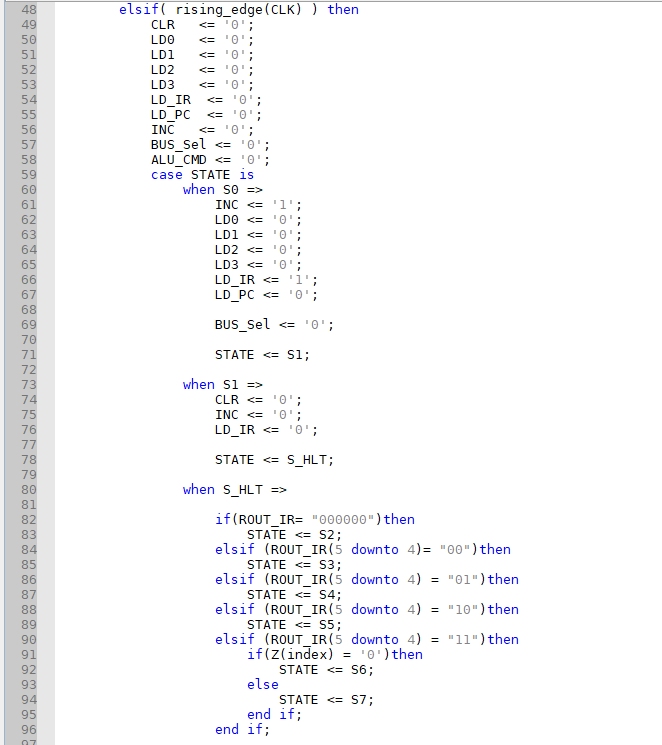
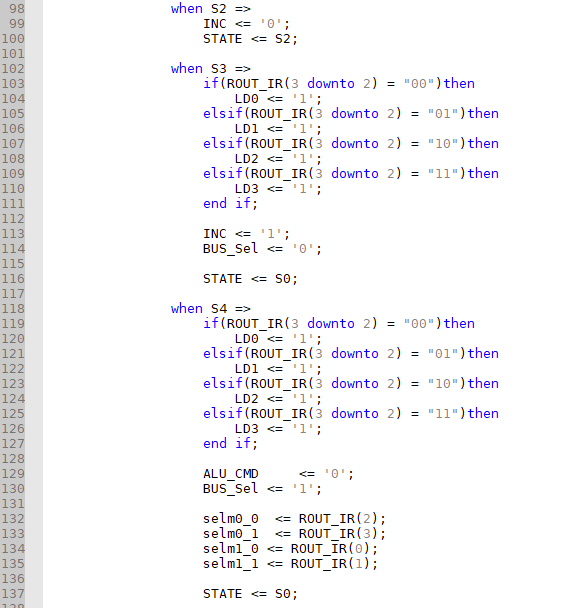
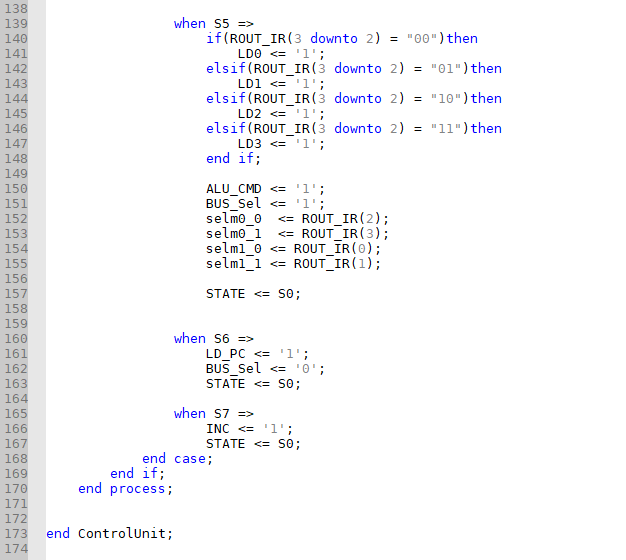
رجیستر Program Counter که وظیفه اشاره به آدرس در حافظه و ارسال آن به واحد کنترل را برعهده دارد

Memory:

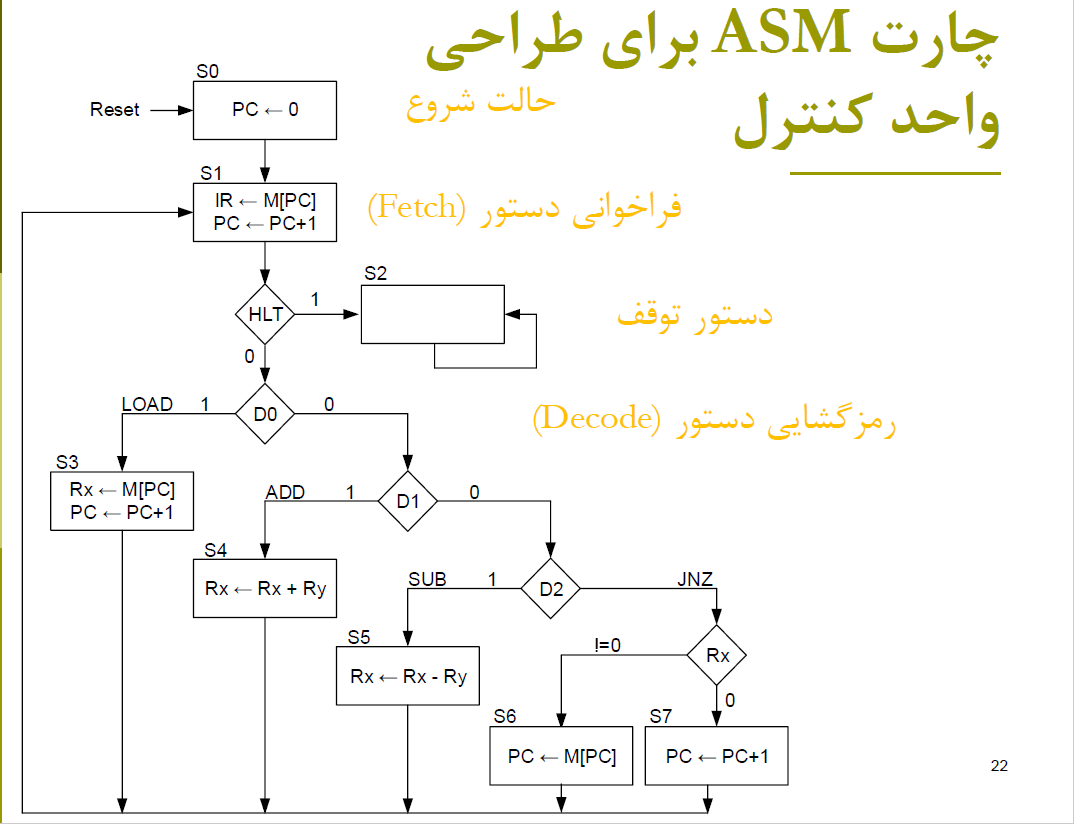


کد برنامه درون مموری ریخته میشود. وظیفه ایجاد این کد بر عهده اسمبلر است که به عنوان نمره اضافی طراحی شده. (اسمبلر در قسمت مربوط به خود توضیح داده میشود) عکس تغییر کنه!!!!!

ControlUnit:

واحد کنترل اصلی ترین بخش پردازنده است. طراحی آن طبق این فلوچارت انجام شده:



در استیت 0، تمام مقادیر را 0 میکنیم و مقدار PC را ریست میکنیم.

در استیت 1، به IR دستور خواندن میدهیم و PC را یک واحد جلو میبریم

در استیت hlt، دستور را از IR میگیریم و بر اساس opcode تصمیم میگیریم چکار کنیم

در استیت 2، دستور را متوقف میکنیم

در استیت 3، رجیستر مبدا را لود میکنیم و PC را رو به جلو میبریم

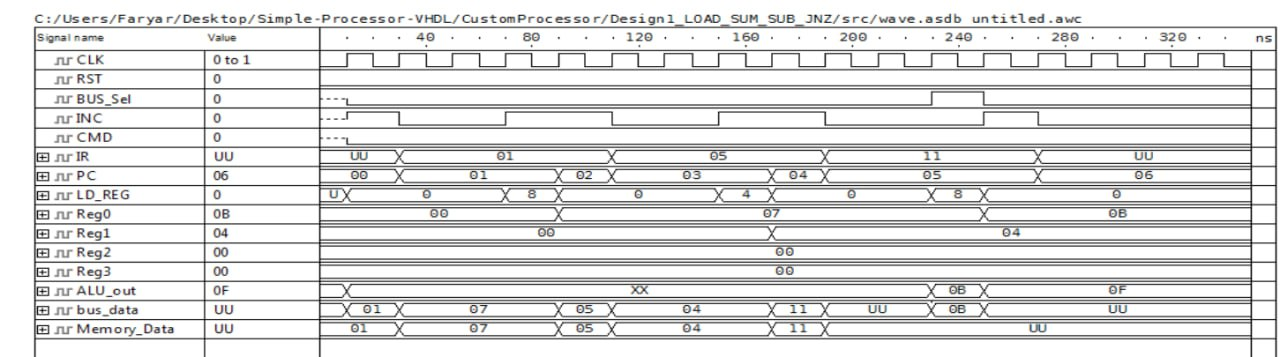
در استیت 4، مقادیر select را برای ALU تعیین میکنیم و رجیستر مبدا را لود میکنیم و همچنین عمل جمع را مشخص میکنیم

در استیت 5، مشابه استیت 4 است ولی برای عمل تفریق

در استیت 6، PC را به آدرس مقصد عمل JNZ منتقل میکنیم

در استیت 7، PC را به جلو میبریم.

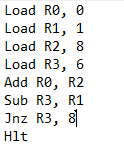
حال بررسی اجرا برای عمل جمع:



مقدار R0=0BHex را مشاهده میکنیم که برابر با 11 است. پس به درستی عمل میکند.

بخش دوم: طراحی ضرب به صورت نرم افزاری

برای این کار میبایست عمل جمع را به تعداد کافی انجام دهیم، در این صورت، 8\*6 باید شش بار عدد 8 را با خودش جمع کنیم، این کار را با دستورات ADD و JNZ انجام میدهیم:



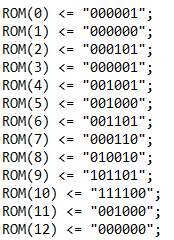
عمکرد به این صورت است: ابتدا مقادیر 0 و 1 در رجیستر لود میشوند،

عمل جمع شدن در رجیستر 0 انجام میشود و از رجیستر 1 برای کم کردن طرف دوم ضرب، در اینجا عدد 6، استفاده میشود. هربار که رجیستر 0 را با عدد 8 جمع کردیم، یک واحد از رجیستر 3 که عدد 6 است کم میکنیم.

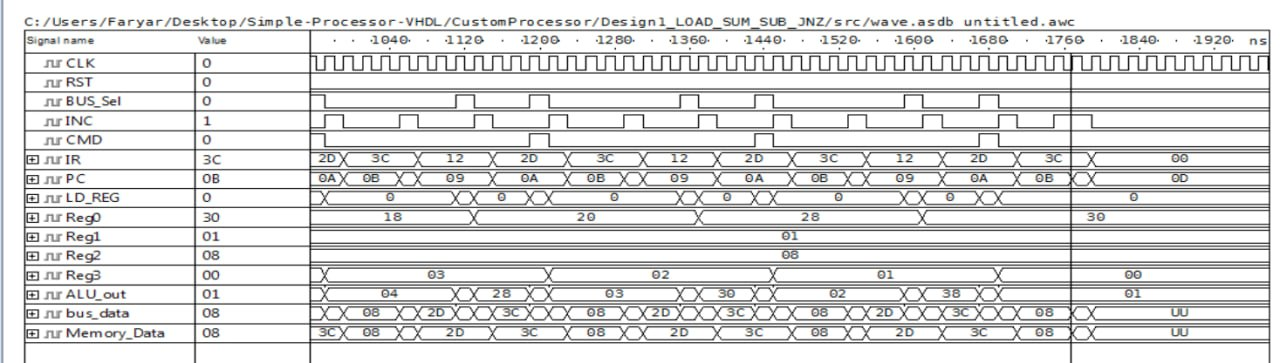
این کار را آنقدر تکرار میکنیم تا مقدار رجیستر 3 به 0 برسد.

در این حالت، 6 بار جمع انجام شده و رجیستر 0 عدد 48 نشان میدهد.

با اسمبل کردن کد توسط اسمبلری که به عنوان نمره مثبت طراحی شده (خود اسمبلر در بخش مربوطه توضیح داده میشود) این مقادر در فایل رام ایجاد میشود:



نتیجه پیاده سازی:



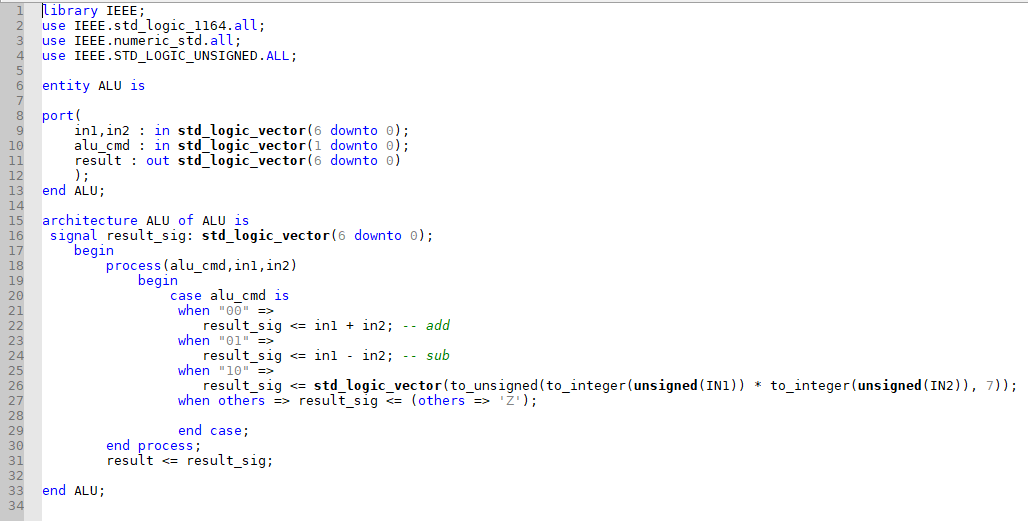
مشاهده میشود که مقدار R0=30Hex که برابر 48 است میشود.

پس پیاده سازی موفق بود.

بخش سوم، سخت افزار ضرب:

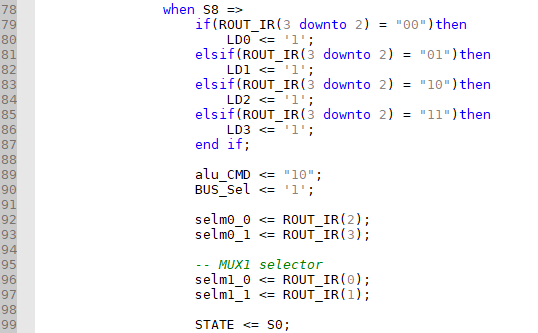
Design2 تقریبا مشابه design1 است ولی در چند کامپوننت تفاوت دارند

ALU:



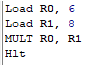
در ازای CMD=00 عمل جمع، CMD=01 هم تفریق، CMD=10 عمل ضرب را انجام میدهد.

ControlUnit:

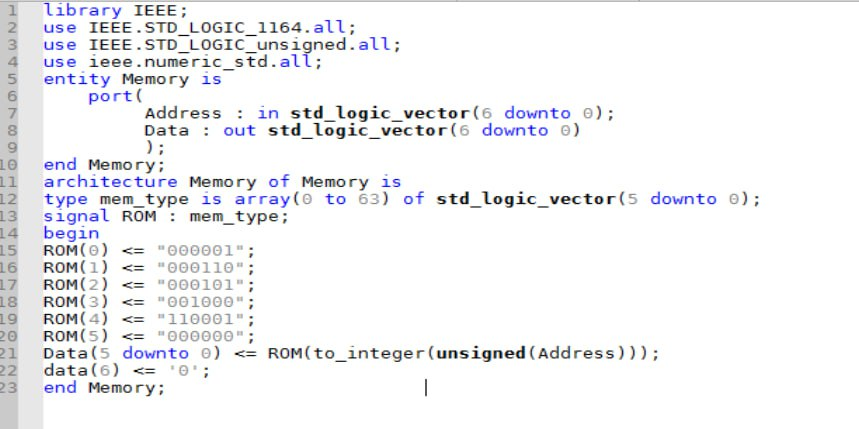


مشابه design1 است ولی استیت هشتم اضافه شده.

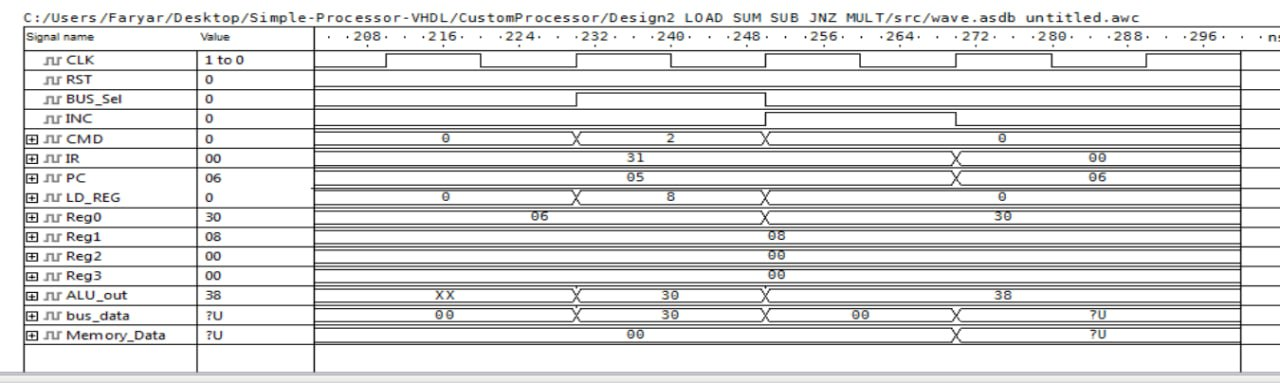
حال باید ضرب را تست کنیم:



کد اسمبلی آن به این صورت است. با دادن این کد به اسمبلر design2، فایل مموری به این شکل میشود:



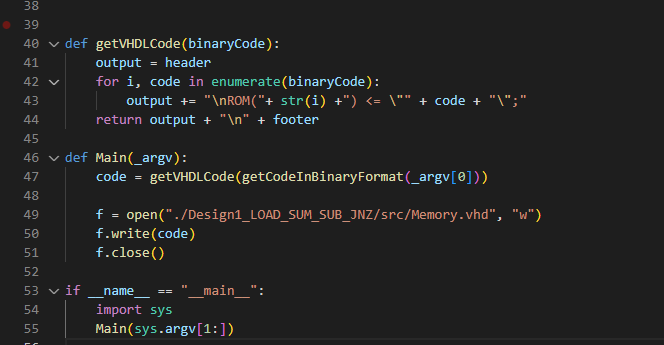
حال باید تست بگیریم:

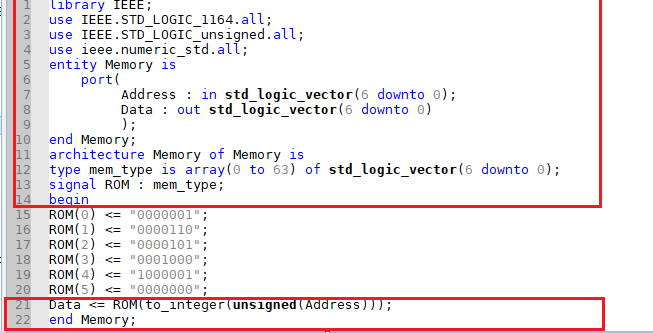


مشاهده میشود که به درستی عمل میکند.

اسمبلر، بخش نمره اضافی:

اسمبلر بخش 1و 2 از design1:

 دو بخش هدر و فوتر، نوشتن قالب ثابت فایل رام هستند که تغییر نمیکنند. یعنی بخش های مشخص شده را مینویسنند:



اصل کار بخش میانی است که باید برنامه نویسی شود

برای دستور LOAD، ابتدا opcode=00، سپس رجیستر مبدا، و دو رقم آخر 01 طراحی میشود. در خانه بعدی حافظه نیز، مقداری که باید لود شود نوشته میشود:



برای دستور ADD، ابتدا opcode =01، سپس رجیستر مبدا و مقصد نوشته میشوند:



برای دستور SUB، ابتدا opcode = 10، سپس رجیستر مبدا و مقصد نوشته میشوند:

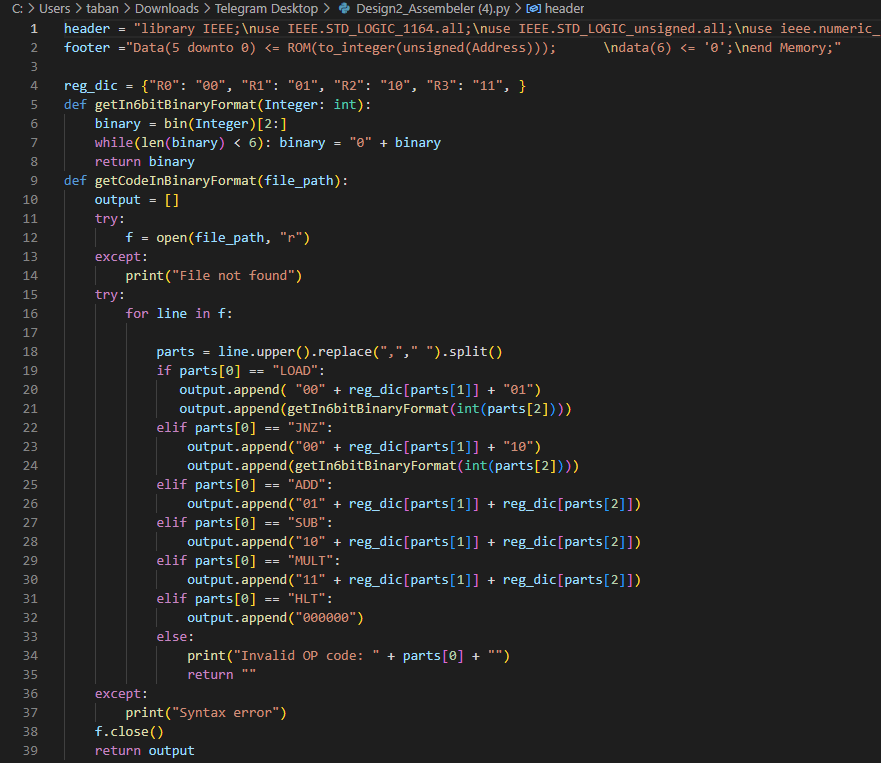


برای دستور JNZ، ابتدا opcode = 11، سپس رجیستر مبدا، و دو رقم راست 00 طراحی میشوند، در خط بعدی نیز آدرس پرش نوشته میشود:



دستور HLT با کد 000000 نمایش داده میشود:



اسمبلر بخش 3 از design2: مشابه 1 است ولی opcode ها تغییر کردند.  

برای دستور LOAD، ابتدا opcode=00، سپس رجیستر مبدا، و دو رقم آخر 01 طراحی میشود. در خانه بعدی حافظه نیز، مقداری که باید لود شود نوشته میشود:



برای دستور ADD، ابتدا opcode =01، سپس رجیستر مبدا و مقصد نوشته میشوند:



برای دستور SUB، ابتدا opcode = 10، سپس رجیستر مبدا و مقصد نوشته میشوند:



برای دستور JNZ، ابتدا opcode = 00، سپس رجیستر مبدا، و دو رقم راست 10 طراحی میشوند، در خط بعدی نیز آدرس پرش نوشته میشود:



دستور HLT با کد 000000 نمایش داده میشود:



برای دستور MULT، ابتدا opcode = 11، سپس رجیستر های مبدا و مقصد نوشته میشوند.

